NATIONAL WEATHER SERVICE
OFFICE of HYDROLOGIC DEVELOPMENT

SCIENTIFIC ALGORITHM DOCUMENTATION STANDARD

Prepared by:
National Oceanic and Atmospheric Administration
National Weather Service
Office of Hydrologic Development
1325 East West Highway
Silver Spring, MD 20910

Table of Contents

Purpose of this Document

This document contains guidelines for writing scientific documentation of algorithms for everyone working in the Hydrology Laboratory (HL). Its purpose is to describe a standard method for articulating scientific algorithms in such a way as to be understood by all interested parties.

All scientific algorithms documented after the approval date of this Scientific Documentation Standard, whether documenting new algorithms or existing prototypes or pieces of software, shall follow this standard.

Scientific documentation will serve as a resource for scientists as well as for software engineers. For scientists, it will serve as a description of one possible implementation of a scientific algorithm as well as a historical record of the steps required to realize that particular implementation. This record should prove valuable over time, especially as research projects are passed from one scientist to another.

For the software engineer, scientific documentation will facilitate translation from scientific theory to a procedural representation in non-scientific terms. It will also serve as a historic record, documenting the steps taken to translate a given scientific algorithm into a programming language.

The guidelines contained here are applicable in two different situations:

1. When describing an existing algorithm (i.e. an existing piece of prototypical or operational software)
2. When describing an algorithm that has not yet been implemented in software

These scientific documents must be maintained in the same fashion as the software used to realize them. Whenever modifications are made to the software described by a scientific document, it is the responsibility of the software engineer to ensure that either

1. No software modifications have altered the fundamental "science" of any given algorithm, or
2. If a science algorithm is modified, the scientific document is also changed to reflect the changes made to the software and, most importantly, the changes to the scientific document are revalidated by the Hydrologic Science and Modeling Branch (HSMB) to ensure that the scientific integrity of the algorithm is preserved.

This does not mean that software engineers should never alter a scientific algorithm. On the contrary, HSEB personnel should always strive to find the most efficient, clear, and maintainable software realization for any algorithm. However, this should only be done in such a way as to preserve the principles and theory contained in the scientific documentation. Software engineers must work closely with scientists to ensure that scientific theory is preserved throughout all stages of development.

1. Document History

This section is applicable in all cases.

Each scientific document will have a section used to maintain a historical record of changes made to the document. It should record, at a minimum, the name of the original author, the creation date of the document's first draft, and a single entry for each noteworthy modification made to the document. An example is included below:

Revision History
(Example)

Date	Version	Description	Author
1 Feb 2006	1.0	Preliminary version of scientific theory	Albert Einstein
6 Jun 2006	1.1	Wrote software implementation	Alan Turing
13 Dec 2006	2.0	Added Data Flow Diagrams; optimized algorithm; added object-oriented structure, software classes	Bill Gates

2. Introduction

This section is applicable in all cases.

The introduction section will include four subsections, *Executive Summary*, *Terminology*, *Background*, and *Problem Statement*. Each subsection should be brief. The purpose here is to introduce the problem the algorithm solves and outline the scientific background by giving an overview of the algorithm. It will be important in the early introduction of the document to clearly inform the reader on the intent of the algorithm as well as the problem the algorithm is addressing. The author of the scientific document should keep in mind that this document is not a technical paper and should be written in such a way that the software engineers can understand. The intended readers are software engineers who may not have specific domain knowledge.

2.1 Executive Summary

The executive summary is a summary of extant work and a statement of the author's specific scientific algorithm to be explained. Executive summaries are written for someone who most likely does not have time to read the entire document. The executive summary gives the reader the essential contents of the author's document. It previews the main points of the document, enabling readers to establish a basis from which they will build upon and for understanding the detailed information contained in the document. It will additionally help readers determine the key scientific results and recommendations reported in the document.

2.2 Terminology

The role of terminology is important and it defines concepts used in the scientific document. The definitions should be made precise, concise, and unambiguous. Each term should be used in one and only one way throughout the document.

2.3 Background

The background section provides an overview and history of the problem.

2.4 Problem Statement

The statement of the problem should include a clear statement outlining the problem and why the problem is important and needs to be addressed. It should contain a sketch of the problem/situation and the background history that led to the development of the algorithm, including why the problem was investigated and how it relates to earlier work that has been done in the field. It should show that the author thoroughly understands the problem.

3. Scientific Description

This section is applicable in all cases.

This section will describe the scientific algorithm in brief. It gives the summary or abstract of the algorithm in a scientific way such that the reader can easily follow the detailed description of the algorithm in the following subsections.

3.1 Methodology or Theory – (equations and computations)

This subsection is used to provide the methodology or theory of the scientific algorithm in detail. The relevant research results need to be included as well. The description does not need to be as detailed as a peer-review journal paper. However, the contents, which are necessary for understanding specifics of the algorithm, especially for translating it into software, must be included. Required equations and computations should be described and explained term by term. Further subsections can be added if needed under this subsection. There is no standard format to be followed for completing this subsection. Nevertheless, it is recommended that all equations, symbols, etc. should follow the popular standard of peer-reviewed journals (e.g. AMS journals). In general, the methodology or theory should be easily understandable, logically clear, and closely related to section 5. Algorithm .

3.2 Limitations

The Limitations section will describe prerequisites for the algorithm. These include limitations caused by the input data, special types of data required, limitations inherent in the technique, which the algorithm is based on, situations where the algorithm cannot be applied, potential implementation difficulties, and any other problems known to be associated with the algorithm. This section will also describe both the implementation and operational impacts of these limitations.

3.3 Future Work

This subsection is used to describe future work about the scientific algorithm which is ongoing, planned, or projected in the near future within OHD or through OHD collaboration with another office. The future work defined here will highlight important considerations in the design of the software to accommodate possible extensions or future modifications.

4. Use Cases

This section is only applicable when an existing prototype is being documented.

Use Cases are a way to express software requirements. They bridge the gap between user needs and system functionality by directly stating the user's intention and system response for each step in a particular interaction. No single Use Case specifies the entire requirements of the system. A Use Case itself is an interaction that a User or another System has with the system that is being designed in order to achieve a goal. Different scenarios within a Use Case show how the goal succeeds or fails. A success scenario is one in which the goal is achieved; a failure scenario is one where the goal is not achieved. Because the goals summarize the intention of the various uses of the system, the users can see how they are supposed to use the system. Users can also spot when the system does not support all of their goals without having to wait for the first prototype or having to wait for the system to be developed.

This is a summary of how to write Use Cases.

1. The name of the use case should start with a strong verb.
2. A Use Case is a set of scenarios. A scenario is a list of steps.
3. Each step should state what the user does and/or how the system responds.
4. The steps must not mention how the system does something.
5. Each step needs to be analyzed in detail before it becomes code.
6. Keep It Simple: use the simplest format you need.
7. Keep track of different versions.
8. Writing use cases is a team sport.
9. Focus on a particular user (give them a name).
10. Don't get bogged down in all the special ways it can go wrong until you've finished the main success story.

4.1 Example Use Case

Taken from *Operational Requirements Document, Operational Implementation of a Distributed Hydrologic Model, (DHM) Build 1 AWIPS OB7*

Use Case Example	**Analyzes Model input and output time series data (forecast and calibration mode)**
Date created:	09/08/05
Actor:	Research Scientist, RFC Hydrologic Forecaster
Description:	DHM will use existing software to analyze input and output time series data

Preconditions:	User must have previously ran Model
Postconditions:	User must be satisfied with results and should also have knowledge on how to edit model data
Priority:	High
Frequency of use:	Daily or as often as every hour
Normal course:	Forecast Mode: Use existing procedures to get IFP to display input/output time series data (e.g. a PLOT-TUL or PLOTTS operation has been previously defined to show the desired data)
Alternative course:	Calibration Mode: Use existing procedures to get ICP to display input/output time series data (e.g. a PLOTTS operation has been previously defined to show the desired data), or use XDMS.
Exceptions:	
Assumptions:	User must have familiarity with the existing river forecasting operations workflow
Notes and Issues:	

5. Algorithm

With the exception of Section 5.7 Algorithm Description, the instructions provided in this section of the document only need to be included when documenting an existing prototype (i.e. they are not necessary when only capturing science).

5.1 System Dependencies

This section is only applicable when an existing prototype is being documented.

This section will provide a description of system dependencies, constrains, and assumptions regarding the algorithm/model and its use. These may be concerning such issues as:

- Assumptions based on other algorithms or models
- End-user characteristics
- Possible and/or probable changes in functionality
- Necessary conditions for the algorithm

General Constraints

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is *not* exhaustive):

- Hardware or software environment
- End-user environment
- Availability or volatility of resources

- Standards compliance
- Interoperability requirements
- Interface/protocol requirements
- Data repository and distribution requirements
- Security requirements (or other such regulations)
- Memory and other capacity limitations
- Performance requirements
- Network communications
- Verification and validation requirements (testing)
- Other means of addressing quality goals
- Other requirements described in the requirements specification

5.2 Data Dictionary

This section is only applicable when an existing prototype is being documented.

The Data Dictionaries shall compile data elements in two ways. "Appendix C – Data Elements List, Alphabetical Order" shall list all the data elements in the Scientific Document in alphabetical order. "Appendix D – Data Elements List, Subroutine Order" shall list the data elements for each subroutine or sub-algorithm in alphabetical order. Each listing should include the following:

1. Variables – all input, output, and processing (local) variables with a description of each variable's data type (e.g. floating point, integer, character string), precision, units (e.g. degrees, percentages), valid ranges (e.g. 0 – infinity; -1.0 – +1.0 inclusive), string length for character strings, default values if any, and a short description of the information the variable contains (e.g. "Width of the radar beam in degrees …").
2. File names – whether the file is used for input or output, a brief description of the data, and the format of the data (or a reference to an interface control document (ICD))
3. Subroutines – a brief description of the subroutine, including mandatory and optional input and output data elements, usage, and function performed.

All data element names should be self-descriptive (i.e. the name should describe what the data element contains or, in the case of subroutines, the function that it performs) and, with the possible exception of loop counters, should never be single characters. Widely recognized symbols from textbooks and professional literature (e.g., DBZE for Reflectivity Factor) are acceptable.

5.3 Data Element Examples

This section is only applicable when an existing prototype is being documented.

Name:	Beam_Blockage_Percent (Output)
Type:	Floating point
Units:	Percentages
Range:	0.000 – 1.000 (i.e. 0% – 100%)
Precision:	0.001 (i.e. 0.1%)

Description: Describes the total percent of the current elevation scan that is obscured due to buildings, terrain, or other geographical features. It is computed within this routine by dividing the number of unobscured radials in the current elevation scan (Num_Unobscured_Radials) by the total number of radials in the current elevation scan (Num_Total_Radials).

Name: Bins_processed (Processing)
Type: Integer
Units: N/A
Description: A local variable used to keep track of the number of sample bins processed so far.

Name: Calc_Beam_Blockage
Type: Subroutine
Calling subroutine: Calc_Precip_Rate
External subroutines called: None
Input parameters: Num_Unobscured_Radials, Num_Total_Radials
Output parameter: Beam_Blockage_Percent
Usage: Calc_Beam_Blockage (Num_Unobscured_Radials, Num_Total_Radials, Beam_Blockage_Percent)
Description: This subroutine will calculate the beam blockage percent for a given elevation scan by dividing the number of unobscured radials in the elevation scan (Num_Unobscured_Radials) by the total number of radials in the elevation scan (Num_Total_Radials).

Name: Precipitation_Rate (Input/Output)
Type: Floating point
Size: 4 bytes[1]
Units: millimeters/hour (mm/hr)
Range: 0.00 – 100.00
Precision: 0.01 mm/hour
Description: Contains the most current precipitation rate. This variable is input to this routine where it is first written to the log file "precip_rate.out" in binary format. It is then passed to the routine "Calculate_Precip_Rate" where it is updated.

Name: precip_rate.out
Type: Output file
Description: This file contains a running history of precipitation rates, stored every 15 minutes for the last 7 days. The format of the file is as follows: each "packet" of binary data is 8 bytes long. The first 4 bytes are a time stamp, in integer format, which represent the number of seconds since midnight, January 1, 1970. The second 4 bytes contain the precipitation rate in floating-point format (see Precipitation_Rate).

5.4 Diagrams (DFDs, Logic Diagrams, State Diagrams, Control Hierarchy, etc.)

This section is only applicable when an existing prototype is being documented.

This section will contain Data flow diagrams (DFDs) which capture the data flow within an algorithm, and is only applicable if there is an existing prototype.

DFDs reveal relationships among and between the various components in a program or system. DFDs help system designers and others during initial analysis stages to visualize a current system or one that may be necessary to meet new requirements. DFDs represent the following:

- External devices sending and receiving data
- Processes that change that data
- Data storage locations

Where applicable, this section should also contain Logic Diagrams and/or State Diagrams. Logic Diagrams represent logical concepts and State Diagrams represent the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

5.5 Other Applicable Standards

This section is only applicable when an existing prototype is being documented.

This section shall describe by reference any applicable standards, and the specific sections of any such standards, which apply to the algorithm being described.

For example, this could include OHD General Software Development Standards and Guidelines, etc.

5.6 Interface requirements and references (e.g. ICDs)

This section is only applicable when an existing prototype is being documented.

This section shall describe how the algorithm interfaces with other algorithm products or users for input and output. Examples of such interfaces include library routines and data streams. This section may also reference external documents that describe interface formats in detail (e.g. Interface Control Documents or ICDs). This section will also contain a list of references to all related data, standards, and technical sources used in algorithm development.

5.7 Algorithm Description

This section is applicable in all cases.

The Algorithm Description portion of a Scientific Document will consists of two sections: Section 5.7.1 Computations, which provides mathematical formulas for the instructions in the Procedures section, and Section 5.7.2 Procedures (Pseudocode), which describes the processing functions in a step-by-step fashion using pseudocode. Section 5.7.3 Pseudocode Constructs below lists the relevant constructs to be used in the pseudocode portion.

Note: Section 5.7.3 is not part of a scientific document and is only included here for reference.

5.7.1 Computations

The Computations section will list the mathematical formulas being carried out by the pseudocode in the Procedures section. These formulas will be grouped and labeled by the subroutines where they are implemented. Each formula will be followed by a brief explanation of the symbology used and the method used to realize the implementation in the Procedure section.

5.7.2 Procedures (Pseudocode)

The term "pseudocode" is used here to describe an English-like language that articulates the steps carried out in an algorithm. It allows the author to focus on the logic of the algorithm without being distracted by details of any given programming language's syntax. For the sake of completeness and consistency, pseudocode should follow a general syntax convention (described below). Pseudocode can be augmented with natural language where convenient (e.g. for trivial operations such as swapping two variables).

As an example of pseudocode, the NEXRAD Joint System Program Office (JSPO) has an established standard for pseudocode and for algorithmic documentation called Algorithm Enunciation Language (AEL). When the JSPO created AEL, they had three requirements in mind:

1. First, the AEL had to be implementation independent. In other words, it needed to steer away from the details of software design, software programming steps, data structures, and internal modes of computation. This approach frees the software engineers to select the design, data structure, and internal modes of computation most suited to the WSR-88D System.

2. Second, the AEL had to provide functional requirements. In other words, it must tell what functions an algorithm performs instead of how it performs them.

3. Third, the AEL had to be easy to learn. This requirement is supported by keeping the AEL it defines simple and by mimicking features of languages users may already be familiar with (e.g., FORTRAN 77, C, and ADA).

A four-part description of AEL can be found at S:\OHD-11\NEXRADGroup\Standards, Guidelines, Templates, etc\Standards & Guidelines\Algorithm Descriptions (AEL).

5.7.3 Pseudocode Constructs

This section contains examples of pseudocode constructs to be used in describing a scientific algorithm. The convention is for keywords and commands to appear in uppercase, which variables and procedure names are either lowercase or have the first letter capitalized.

5.7.3.1 Algorithm Construct

The format of an algorithm construct shall be as follows:

```
BEGIN ALGORITHM (algorithm-name)
    Pseudocode statement(s)
    …
    Pseudocode statement(s)
END ALGORITHM (algorithm-name)
```

Example:

```
BEGIN ALGORITHM (Storm Segments)
    COMPUTE (Storm Segment)
END ALGORITHM (Storm Segments)
```

5.7.3.2 DO WHILE Construct

The second DO construct is DO WHILE. The format shall be as follows:

```
DO WHILE (predicate)
    Pseudocode statement(s)
END DO
```

The DO WHILE construct represents a loop of repeated consideration of the pseudocode statement while the value of the predicate is true.

Example:

```
DO WHILE (Reflectivity is above threshold)
    Add the current SAMPLE VOLUME to current segment.
    Update end SAMPLE VOLUME number.
    COMPUTE (maximum Reflectivity Factor (dbze))
    WRITE (maximum Reflectivity Factor (dbze))
END DO
```

5.7.3.3 DO UNTIL Construct

The third DO construct is DO UNTIL. The format shall be as follows:

```
DO UNTIL (predicate)
    Pseudocode statement(s)
END DO
```

This construct represents a loop of repeated consideration of Pseudocode statement until the value of the predicate is true. Pseudocode statement is considered at least once even if the predicate is true at the onset.

Example:

```
DO UNTIL (No more storms isolated)
    Correlate the cells along vertical.
    Find 3D characteristics of correlated cells.
    Add correlated cells as a new storm to the table of storms.
END DO
```

5.7.3.4 IF-THEN-ELSE Construct

The IF-THEN-ELSE construct shall have the following format:

```
IF (predicate) THEN
    Pseudocode statement(s)$_1$
ELSE <optional>
    Pseudocode statement(s)$_2$
END IF
```

If the predicate is true, Pseudocode statement(s)$_1$ (the “THEN” portion) will execute. If the predicate is false, then Pseudocode statement(s)$_2$ (the “ELSE” portion) will execute. Note that the “ELSE” portion of this construct is optional and may not be used in every instance.

Example:

```
IF (REFLECTIVITY is greater than 35 dBZe) THEN
    Label current segment
    COMPUTE (maximum Reflectivity Factor (Reflectivity))
ELSE
    Ignore current segment
END IF
```

5.7.3.5 CASE Construct

The CASE construct shall have the following format:

```
BEGIN CASE (variable-name)
    CASE (value-list-l)
        Pseudocode statement(s)$_1$
    CASE (value-list-2)
```

```
        Pseudocode statement(s)$_2$
        …
    CASE (value-list-n)
        Pseudocode statement(s)$_n$
END CASE
```

where "value-list-*i*" is a series of values "variable-name" can take. Pseudocode statement(s)$_i$ is performed if a variable-name has the same value as a member of value-list-*i*. Example:

```
BEGIN CASE (elapsed time in minutes)
    CASE (6, 12, 18, 24)
        Collect raw radar data
    CASE (7, 13, 19, 25)
        Preprocess raw data
    CASE (8, 14, 20, 26)
        Analyze preprocessed data
    CASE (9, 15, 21, 27)
        Track and forecast the storms
        COMPUTE (Storm Speed)
END CASE
```

5.7.3.6 Logical Constructs

In creating pseudocode, it is often useful to use the logical operators AND, OR, and NOT. The format for using these shall be as follows:

- (predicate$_1$) AND (predicate$_2$)
- (predicate$_1$) OR (predicate$_2$)
- NOT (predicate)

Example:

```
IF (Velocity is greater than 10 knots) AND (Velocity is less than 20 knots) THEN
    Label current segment
ELSE
    Ignore current segment
END IF
```

5.7.3.7 I/O Constructs

The READ and WRITE constructs indicate when the algorithm will input or output data items and shall have the following formats:

```
READ (variable-name)
```

and:

WRITE (variable-name)

Examples:

READ (Maximum Reflectivity Factor (dbze))

and:

WRITE (Centroid Position)

5.7.3.8 COMPUTE and EXIT Constructs

5.7.3.8.1 COMPUTE Construct

The COMPUTE construct is a shorthand notation for a set of computations used to derive a value or set of values. The COMPUTE construct shall have the following format:

COMPUTE (variable-name)

This construct causes the value(s) for variable-name to be computed. Example:

COMPUTE (Minimum Velocity (Doppler))

causes the algorithm to compute the minimum Doppler velocity.

5.7.3.8.2 EXIT Construct

The EXIT construct shall have the following format:

EXIT ALGORITHM (algorithm-name)

This key word forces an unconditional exit from the body of the algorithm, terminating its execution.

6. Outputs (format, real time, rate, etc.)

This section is applicable in all cases.

This section of the document should contain a description of all outputs produced or modified by the activities in this algorithm. It should describe the format of the output data in sufficient detail for downstream algorithms to be easily interfaced, stating such factors as whether the data should be output in real time (and if so, how often), the format of the data, and the destination of the output (e.g. file, other process, graphical user interface, etc.).

7. References

This section is applicable in all cases.

This section shall provide all the references which appear in this document. They need to be in the standard format like: author(s), year, title, title of journal, volume number, page number, etc. They need to be in alphabetical order based on the last name of the first author.

Appendices

Appendices contain a collection of useful information that does not need to be included in the sections that describe the information. Appendices may contain tables, lists, calculations, data, background information, and other types of information.

Appendices A – D should always be the same (i.e. Appendix A should always contain the Acronym List, Appendix B should always be the Glossary, etc.). Use appendices E and beyond for additional information as necessary.

Appendix A – Acronym List

This section is applicable in all cases.

This appendix will contain an alphabetical listing of acronyms used within the document. A sample listing of common acronyms follows:

AEL	Algorithm Enunciation Language
AP	Anomalous Propagation
AWIPS	Advanced Weather Interactive Processing System
DHR	Digital Hybrid Scan Reflectivity
DPA	Hourly Digital Precip Array
DSP	Digital Storm Total Precip
ECP	Engineering Change Proposal
HL	Hydrology Laboratory
HSEB	Hydrologic Software Engineering Branch
HSMB	Hydrologic Science and Modeling Branch
HSR	Hybrid Scan Reflectivity
NCDC	National Climatic Data Center
NEXRAD	NEXt Generation RADar
NWS	National Weather Service
OHD	Office of Hydrologic Development
OHP	One Hour Precip (1 Hr Surface Rainfall Accumulation)
RDA	Radar Data Acquisition
ROC	Radar Operations Center
RPG	Radar Products Generator
VCP	Volume Coverage Pattern

Appendix B – Glossary

This section is applicable in all cases.

This appendix will contain an alphabetical listing of terms used within the document as well as their definitions. A sample glossary follows:

Algorithm Enunciation Language (AEL) – A structured description/specification language that closely resembles high-level structured computer languages.

Engineering Change Proposal (ECP) – A formal proposal to change the WSR-88D's configuration.

Next Generation Weather Radar (NEXRAD) – A program that will improve the nation's Doppler radar weather coverage by utilizing improved hardware and software for weather forecasting.

Appendix C – Data Elements List, Alphabetical Order

This section is applicable in all cases.

This appendix will contain an alphabetical listing of all data elements used within the document as well as their descriptions. See section 5.2 Data Dictionary for examples.

Appendix D – Data Elements List, Subroutine Order

This section is applicable in all cases.

This appendix will contain a listing of all data elements used within the document as well as their descriptions, similar to and duplicating the listing in Appendix C – Data Elements List, Alphabetical Order. However, this appendix will group and label data element entries by subroutine or sub-algorithm. See section 5.2 Data Dictionary for examples.